N 3

```
EEEEEEEEEE  XX      XX    AAAAAA    MM      MM  PPPPPPPP    LL          EEEEEEEEEE    SSSSSSSS
EEEEEEEEEE  XX      XX    AAAAAA    MM      MM  PPPPPPPP    LL          EEEEEEEEEE    SSSSSSSS
EEEEEEEEEE  XX      XX    AAAAAA    MM      MM  PPPPPPPP    LL          EEEEEEEEEE    SSSSSSSS
EE          XX      XX  AA      AA  MMMM  MMMM  PP      PP  LL          EE          SS
EE          XX      XX  AA      AA  MMMM  MMMM  PP      PP  LL          EE          SS
EE          XX      XX  AA      AA  MMMM  MMMM  PP      PP  LL          EE          SS
EE            XX  XX    AA      AA  MM  MM  MM  PP      PP  LL          EE          SS
EE            XX  XX    AA      AA  MM  MM  MM  PP      PP  LL          EE          SS
EE            XX  XX    AA      AA  MM  MM  MM  PP      PP  LL          EE          SS
EEEEEEEE        XX      AA      AA  MM      MM  PPPPPPPP    LL          EEEEEEEE      SSSSSS
EEEEEEEE        XX      AA      AA  MM      MM  PPPPPPPP    LL          EEEEEEEE      SSSSSS
EEEEEEEE        XX      AA      AA  MM      MM  PPPPPPPP    LL          EEEEEEEE      SSSSSS
EE            XX  XX    AAAAAAAAAA  MM      MM  PP          LL          EE                  SS
EE            XX  XX    AAAAAAAAAA  MM      MM  PP          LL          EE                  SS
EE            XX  XX    AAAAAAAAAA  MM      MM  PP          LL          EE                  SS
EE          XX      XX  AA      AA  MM      MM  PP          LL          EE                  SS
EE          XX      XX  AA      AA  MM      MM  PP          LL          EE                  SS
EE          XX      XX  AA      AA  MM      MM  PP          LL          EE                  SS
EEEEEEEEEE  XX      XX  AA      AA  MM      MM  PP          LLLLLLLLLL  EEEEEEEEEE  SSSSSSSS
EEEEEEEEEE  XX      XX  AA      AA  MM      MM  PP          LLLLLLLLLL  EEEEEEEEEE  SSSSSSSS
EEEEEEEEEE  XX      XX  AA      AA  MM      MM  PP          LLLLLLLLLL  EEEEEEEEEE  SSSSSSSS
```

LPN

FILEID**XADRIVER

```
XX      XX     AAAAAA     DDDDDDDD     RRRRRRRR       IIIIII     VV      VV   EEEEEEEEEE   RRRRRRRR
XX      XX     AAAAAA     DDDDDDDD     RRRRRRRR       IIIIII     VV      VV   EEEEEEEEEE   RRRRRRRR
XX      XX   AA      AA   DD      DD   RR      RR       II       VV      VV   EE           RR      RR
XX      XX   AA      AA   DD      DD   RR      RR       II       VV      VV   EE           RR      RR
  XX  XX     AA      AA   DD      DD   RR      RR       II       VV      VV   EE           RR      RR
  XX  XX     AA      AA   DD      DD   RR      RR       II       VV      VV   EE           RR      RR
    XX       AA      AA   DD      DD   RRRRRRRR         II       VV      VV   EEEEEEEE     RRRRRRRR
    XX       AA      AA   DD      DD   RRRRRRRR         II       VV      VV   EEEEEEEE     RRRRRRRR
  XX  XX     AAAAAAAAAA   DD      DD   RR  RR           II       VV      VV   EE           RR  RR
  XX  XX     AAAAAAAAAA   DD      DD   RR  RR           II       VV      VV   EE           RR  RR
XX      XX   AA      AA   DD      DD   RR    RR         II         VV  VV     EE           RR    RR         ....
XX      XX   AA      AA   DD      DD   RR    RR         II         VV  VV     EE           RR    RR         ....
XX      XX   AA      AA   DDDDDDDD     RR      RR     IIIIII         VV       EEEEEEEEEE   RR      RR       ....
XX      XX   AA      AA   DDDDDDDD     RR      RR     IIIIII         VV       EEEEEEEEEE   RR      RR       ....

MM      MM     AAAAAA     RRRRRRRR
MM      MM     AAAAAA     RRRRRRRR
MMMM  MMMM   AA      AA   RR      RR
MMMM  MMMM   AA      AA   RR      RR
MM  MM  MM   AA      AA   RR      RR
MM  MM  MM   AA      AA   RR      RR
MM      MM   AA      AA   RRRRRRRR
MM      MM   AA      AA   RRRRRRRR
MM      MM   AAAAAAAAAA   RR  RR
MM      MM   AAAAAAAAAA   RR  RR
MM      MM   AA      AA   RR    RR
MM      MM   AA      AA   RR    RR
MM      MM   AA      AA   RR      RR
MM      MM   AA      AA   RR      RR
```

```
XAD

;++
; X
; F



; I

; C

;--

XA_
; R

; F

;
;
;
;

;++
; S
;
```

```
        .TITLE  XADRIVER - VAX/VMS DR11-W DRIVER
        .IDENT  'V04-001'

;
```

```
;
;++
;
; FACILITY:
;
;       VAX/VMS Executive, I/O Drivers
;
; ABSTRACT:
;
;       This module contains the DR11-W driver:
;
;               Tables for loading and dispatching
;               Controller initialization routine
;               FDT routine
;               The start I/O routine
;               The interrupt service routine
;               Device specific Cancel I/O
;               Error logging register dump routine
;
; ENVIRONMENT:
;
;       Kernal Mode, Non-paged
;
; AUTHOR:
;
;       C. A. Sameulson 10-JAN-79
;
; MODIFIED BY:
;
```

```
;
;       V04-001 JLV0395         Jake VanNoy             6-SEP-1984
;               Add AVL bit to DEVCHAR.
;
;       V03-006 TMK0001         Todd M. Katz            07-Dec-1983
;               Fix a broken branch.
;
;       V03-005 JLV0304         Jake VanNoy             24-AUG-1983
;               Several bug fixes. All word writes to XA_CSR now have
;               ATTN set so as to prevent lost interrupts. Attention
;               AST list is synchronized at device IPL in DEL_ATTNAST.
;               Correct status is returned on a set mode ast that
;               is returns through EXE$FINISHIO. REQCOM's are always
;               done at FIPL.  Signed division that prevented full size
;               transfers has been fixed.
;
;
;       V03-004 KDM0059         Kathleen D. Morse       14-Jul-1983
;               Change time-wait loops to use new TIMEDWAIT macro.
;               Add $DEVDEF.
;
;       V03-003 KDM0002         Kathleen D. Morse       28-Jun-1982
;               Added $DYNDEF, $DCDEF, and $SSDEF.
;
;--
```

```
        .SBTTL  External and local symbol definitions

; External symbols

        $ACBDEF                         ; AST control block
        $CRBDEF                         ; Channel request block
        $DCDEF                          ; Device types
        $DDBDEF                         ; Device data block
        $DEVDEF                         ; Device characteristics
        $DPTDEF                         ; Driver prolog table
        $DYNDEF                         ; Dynamic data structure types
        $EMBDEF                         ; EMB offsets
        $IDBDEF                         ; Interrupt data block
        $IODEF                          ; I/O function codes
        $IPLDEF                         ; Hardware IPL definitions
        $IRPDEF                         ; I/O request packet
        $PRDEF                          ; Internal processor registers
        $PRIDEF                         ; Scheduler priority increments
        $SSDEF                          ; System status codes
        $UCBDEF                         ; Unit control block
        $VECDEF                         ; Interrupt vector block
        $XADEF                          ; Define device specific characteristics

; Local symbols

; Argument list (AP) offsets for device-dependent QIO parameters

P1      = 0                             ; First QIO parameter
P2      = 4                             ; Second QIO parameter
P3      = 8                             ; Third QIO parameter
P4      = 12                            ; Fourth QIO parameter
P5      = 16                            ; Fifth QIO parameter
P6      = 20                            ; Sixth QIO parameter

; Other constants

XA_DEF_TIMEOUT  = 10                    ; 10 second default device timeout
XA_DEF_BUFSIZ   = 65535                 ; Default buffer size
XA_RESET_DELAY  = <<2+9>/10>            ; Delay N microseconds after RESET
                                        ;  (rounded up to 10 microsec intervals)

; DR11-W definitions that follow the standard UCB fields
; *** N O T E ***  ORDER OF THESE UCB FIELDS IS ASSUMED

        $DEFINI UCB
        .=UCB$L_DPC+4
$DEF    UCB$L_XA_ATTN                   ; Attention AST listhead
                .BLKL   1
$DEF    UCB$W_XA_CSRTMP                 ; Temporary storage of CSR image
                .BLKW   1
$DEF    UCB$W_XA_BARTMP                 ; Temporary storage of BAR image
                .BLKW   1
$DEF    UCB$W_XA_CSR                    ; Saved CSR on interrupt
                .BLKW   1
$DEF    UCB$W_XA_EIR                    ; Saved EIR on interrupt
```

```
                .BLKW   1
$DEF    UCB$W_XA_IDR                    ; Saved IDR on interrupt
                .BLKW   1
$DEF    UCB$W_XA_BAR                    ; Saved BAR register on interrupt
                .BLKW   1
$DEF    UCB$W_XA_WCR                    ; Saved WCR register on interrupt
                .BLKW   1
$DEF    UCB$W_XA_ERROR                  ; Saved device status flag
                .BLKW   1
$DEF    UCB$L_XA_DPR                    ; Data Path Register contents
                .BLKL   1
$DEF    UCB$L_XA_FMPR                   ; Final Map Register contents
                .BLKL   1
$DEF    UCB$L_XA_PMPR                   ; Previous Map Register contents
                .BLKL   1
$DEF    UCB$W_XA_DPRN                   ; Saved Datapath Register Number
                .BLKW   1               ; And Datapath Parity error flag

; Bit positions for device-dependent status field in UCB

        $VIELD  UCB,0,<-                ; UCB device specific bit definitions
                <ATTNAST,,M>,-          ; ATTN AST requested
                <UNEXPT,,M>,-           ; Unexpected interrupt received
                >
UCB$K_SIZE=.
        $DEFEND UCB

; Device register offsets from CSR address

        $DEFINI XA                      ; Start of DR11-W definitions
$DEF    XA_WCR                          ; Word count
                        .BLKW   1
$DEF    XA_BAR                          ; Buffer address
                        .BLKW   1
$DEF    XA_CSR                          ; Control/status

; Bit positions for device control/status register

        $EQULST XA$K_,,0,1,<-           ; Define CSR FNCT bit values
                <FNCT1,2>-
                <FNCT2,4>-
                <FNCT3,8>-
                <STATUSA,2048>-         ; Define CSR STATUS bit values
                <STATUSB,1024>-
                <STATUSC,512>-
        >

        $VIELD  XA_CSR,0,<-             ; Control/status register
                <GO,,M>,-               ; Start device
                <FNCT,3,M>,-            ; CSR FNCT bits
                <XBA,2,M>,-             ; Extended address bits
                <IE,,M>,-               ; Enable interrupts
                <RDY,,M>,-              ; Device ready for command
                <CYCLE,,M>,-            ; Starts slave transmit
                <STATUS,3,M>,-          ; CSR STATUS bits
                <MAINT,,M>,-                    ; Maintenance bit
                <ATTN,,M>,-                     ; Status from other processor
                <NEX,,M>,-                      ; Nonexistent memory flag
                <ERROR,,M>,-                    ; Error or external interrupt
        >
$DEF    XA_EIR                                  ; Error information register

; Bit positions for error information register

        $VIELD  XA_EIR,0,<-                     ; Error information register
                <REGFLG,,M>,-                   ; Flags whether EIR or CSR is accessed
                <SPARE,7,M>,-                   ; Unused - spare
                <BURST,,M>,-                    ; Burst mode transfer occured
                <DLT,,M>,-                      ; Time-out for successive burst xfer
                <PAR,,M>,-                      ; Parity error during DATI/P
                <ACLO,,M>,-                     ; Power fail on this processor
                <MULTI,,M>,-                    ; Multi-cycle request error
                <ATTN,,M>,-                     ; ATTN - same as in CSR
                <NEX,,M>,-                      ; NEX - same as in CSR
                <ERROR,,M>,-                    ; ERROR - same as in CSR
        >
                .BLKW   1

$DEF    XA_IDR                                  ; Input Data Buffer register
$DEF    XA_ODR                                  ; Output Data Buffer register
                .BLKW   1

        $DEFEND XA                              ; End of DR11-W definitions
        .SBTTL  Device Driver Tables

; Driver prologue table

        DPTAB   -                               ; DPT-creation macro
                END=XA_END,-                    ; End of driver label
                ADAPTER=UBA,-                   ; Adapter type
                FLAGS=DPT$M_SVP,-               ; Allocate system page table
                UCBSIZE=UCB$K_SIZE,-            ; UCB size
                NAME=XADRIVER                   ; Driver name
        DPT_STORE INIT                          ; Start of load
                                                ; initialization table
        DPT_STORE UCB,UCB$B_FIPL,B,8            ; Device fork IPL
        DPT_STORE UCB,UCB$B_DIPL,B,22           ; Device interrupt IPL
        DPT_STORE UCB,UCB$L_DEVCHAR,L,<-        ; Device characteristics
                DEV$M_AVL!-                     ; Available
                DEV$M_RTM!-                     ; Real Time device
                DEV$M_ELG!-                     ; Error Logging enabled
                DEV$M_IDV!-                     ;   input device
                DEV$M_ODV>                      ;   output device
        DPT_STORE UCB,UCB$B_DEVCLASS,B,DC$_REALTIME     ; Device class
        DPT_STORE UCB,UCB$B_DEVTYPE,B,DT$_DR11W ; Device Type
        DPT_STORE UCB,UCB$W_DEVBUFSIZ,W,-       ; Default buffer size
                XA_DEF_BUFSIZ
        DPT_STORE REINIT                        ; Start of reload
                                                ; initialization table
        DPT_STORE DDB,DDB$L_DDT,D,XA$DDT        ; Address of DDT
        DPT_STORE CRB,CRB$L_INTD+4,D,-          ; Address of interrupt
                XA_INTERRUPT                    ; service routine
        DPT_STORE CRB,CRB$L_INTD+VEC$L_INITIAL,-; Address of controller
                D,XA_CONTROL_INIT               ; initialization routine
        DPT_STORE END                           ; End of initialization
                                                ; tables

; Driver dispatch table

        DDTAB   -                               ; DDT-creation macro
                DEVNAM=XA,-                     ; Name of device
                START=XA_START,-                ; Start I/O routine
                FUNCTB=XA_FUNCTABLE,-           ; FDT address
                CANCEL=XA_CANCEL,-              ; Cancel I/O routine
                REGDMP=XA_REGDUMP,-             ; Register dump routine
                DIAGBF=<<13*4>+<<3+5+1>*4>>,-   ; Diagnostic buffer size
                ERLGBF=<<13*4>+<1*4>+<EMB$L_DV_REGSAV>> ; Error log buffer size

;
; Function dispatch table
;
XA_FUNCTABLE:                                   ; FDT for driver
        FUNCTAB ,-                              ; Valid I/O functions
                <READPBLK,READLBLK,READVBLK,WRITEPBLK,WRITELBLK,WRITEVBLK,-
                SETMODE,SETCHAR,SENSEMODE,SENSECHAR>
        FUNCTAB ,                               ; No buffered functions
        FUNCTAB XA_READ_WRITE,-                 ; Device-specific FDT
                <READPBLK,READLBLK,READVBLK,WRITEPBLK,WRITELBLK,WRITEVBLK>
        FUNCTAB +EXE$READ,<READPBLK,READLBLK,READVBLK>
        FUNCTAB +EXE$WRITE,<WRITEPBLK,WRITELBLK,WRITEVBLK>
        FUNCTAB XA_SETMODE,<SETMODE,SETCHAR>
        FUNCTAB +EXE$SENSEMODE,<SENSEMODE,SENSECHAR>
        .SBTTL  XA_CONTROL_INIT, Controller initialization

;++
; XA_CONTROL_INIT, Called when driver is loaded, system is booted, or
; power failure recovery.
;
; Functional Description:
;
;       1) Allocates the direct data path permanently
;       2) Assigns the controller data channel permanently
;       3) Clears the Control and Status Register
;       4) If power recovery, requests device time-out
;
; Inputs:
;
;       R4 = address of CSR
;       R5 = address of IDB
;       R6 = address of DDB
;       R8 = address of CRB
;
; Outputs:
;
;       VEC$V_PATHLOCK bit set in CRB$L_INTD+VEC$B_DATAPATH
;       UCB address placed into IDB$L_OWNER
;
;
;--

XA_CONTROL_INIT:

        MOVL    IDB$L_UCBLST(R5),R0     ; Address of UCB
        MOVL    R0,IDB$L_OWNER(R5)      ; Make permanent controller owner
        BISW    #UCB$M_ONLINE,UCB$W_STS(R0)
                                        ; Set device status "on-line"

; If powerfail has occured and device was active, force device time-out.
; The user can set his own time-out interval for each request.  Time-
; out is forced so a very long time-out period will be short circuited.

        BBS     #UCB$V_POWER,UCB$W_STS(R0),10$
                                        ; Branch if powerfail
        BISB    #VEC$M_PATHLOCK,CRB$L_INTD+VEC$B_DATAPATH(R8)
                                        ; Permanently allocate direct datapath
10$:
        BSBW    XA_DEV_RESET            ; Reset DR11W
        RSB                             ; Done
        .SBTTL  XA_READ_WRITE, FDT for device data transfers

;++
; XA_READ_WRITE, FDT for READLBLK,READVBLK,READPBLK,WRITELBLK,WRITEVBLK,
;                         WRITEPBLK
;
; Functional description:
;
;       1) Rejects QUEUE I/O's with odd transfer count
;       2) Rejects QUEUE I/O's for BLOCK MODE request to UBA Direct Data
;          PATH on odd byte boundary
;       3) Stores request time-out count specified in P3 into IRP
;       4) Stores FNCT bits specified in P4 into IRP
;       5) Stores word to write into ODR from P5 into IRP
;       6) Checks block mode transfers for memory modify access
;
; Inputs:
;
;       R3 = Address of IRP
;       R4 = Address of PCB
;       R5 = Address of UCB
;       R6 = Address of CCB
;       R8 = Address of FDT routine
;       AP = Address of P1
;               P1 = Buffer Address
;               P2 = Buffer size in bytes
;               P3 = Request time-out period (cond'tional on IO$M_TIMED)
;               P4 = Value for CSR FNCT bits (conditional on IO$M_SETFNCT)
;               P5 = Value for ODR (conditional on IO$M_SETFNCT)
;               P6 = Address of Diagnostic Buffer
;
; Outputs:
;
;       R0 = Error status if odd transfer count
;       IRP$L_MEDIA = Time-out count for this request
;       IRP$L_SEGVBN = FNCT bits for DR11-W CSR and ODR image
;
;--

XA_READ_WRITE:
        BLBC    P2(AP),10$              ; Branch if transfer count even
2$:     MOVZWL  #SS$_BADPARAM,R0        ; Set error status code
5$:     JMP     G^EXE$ABORTIO           ; Abort request
10$:    MOVZWL  IRP$W_FUNC(R3),R1       ; Fetch I/O Function code
        MOVL    P3(AP),IRP$L_MEDIA(R3)  ; Set request specific time-out count
        BBS     #IO$V_TIMED,R1,15$      ; Branch if time-out specified
        MOVL    #XA_DEF_TIMEOUT,IRP$L_MEDIA(R3)
                                        ; Else set default timeout value
15$:    BBC     #IO$V_DIAGNOSTIC,R1,20$ ; Branch if not maintenance reqeust
        EXTZV   #IO$V_FCODE,#IO$S_FCODE,R1,R1 ; AND out all function modifiers
        CMPB    #IO$_READPBLK,R1        ; If maintenance function, must be
                                        ; physical I/O read or write
        BEQL    20$
        CMPB    #IO$_WRITEPBLK,R1
        BEQL    20$
        MOVZWL  #SS$_NOPRIV,R0          ; No privilege for operation
        BRB     5$                              ; Abort request
20$:    EXTZV   #0,#3,P4(AP),R0                 ; Get value for FNCT bits
        ASHL    #XA_CSR$V_FNCT,R0,IRP$L_SEGVBN(R3) ; Shift into position for CSR
        MOVW    P5(AP),IRP$L_SEGVBN+2(R3) ; Store ODR value for later

; If this is a block mode transfer, check buffer for modify access
; whether or not the function is read or write.  The DR11-W does
; not decide whether to read or write, the users device does.
; For word mode requests, return to read check or write check.
;
; If this is a BLOCK MODE request and the UBA Direct Data Path is
; in use, check the data buffer address for word alignment.  If buffer
; is not word aligned, reject the request.

        BBS     #IO$V_WORD,IRP$W_FUNC(R3),30$
                                                ; Branch if word mode transfer
        BBS     #XA$V_DATAPATH,UCB$L_DEVDEPEND(R5),25$
                                                ; Branch if Buffered Data Path in use
        BLBS    P1(AP),2$                       ; DDP, branch on bad alignment
25$:    JMP     G^EXE$MODIFY                    ; Checke buffer for modify access
30$:    RSB                                     ; Return
```

```
        .SBTTL  XA_SETMODE, Set Mode, Set characteristics FDT

;++
; XA_SETMODE, FDT routine to process SET MODE and SET CHARACTERISTICS
;
; Functional description:
;
;       If IO$M_ATTNAST modifier is set, queue attention AST for device
;       If IO$M_DATAPATH modifier is set, queue packet.
;       Else, finish I/O.
;
; Inputs:
;
;       R3 = I/O packet address
;       R4 = PCB address
;       R5 = UCB address
;       R6 = CCB address
;       R7 = Function code
;       AP = QIO Paramater list address
;
; Outputs:
;
;       If IO$M_ATTNAST is specified, queue AST on UCB attention AST list.
;       If IO$M_DATAPATH is specified, queue packet to driver.
;       Else, use exec routine to update device characteristics
;
;--

XA_SETMODE:
        MOVZWL  IRP$W_FUNC(R3),R0       ; Get entire function code
        BBC     #IO$V_ATTNAST,R0,20$    ; Branch if not an ATTN AST

; Attention AST request

        PUSHR   #^M<R4,R7>
        MOVAB   UCB$L_XA_ATTN(R5),R7    ; Address of ATTN AST control block list
        JSB     G^COM$SETATTNAST        ; Set up attention AST
        POPR    #^M<R4,R7>
        BLBC    R0,50$                  ; Branch if error
        BISW    #UCB$M_ATTNAST,UCB$W_DEVSTS(R5)
                                        ; Flag ATTN AST expected.
        BBC     #UCB$V_UNEXPT,UCB$W_DEVSTS(R5),10$
                                        ; Deliver AST if unsolicited interrupt
        BSBW    DEL_ATTNAST
10$:    MOVZBL  #SS$_NORMAL,R0          ; Set status
        JMP     G^EXE$FINISHIOC         ; Thats all for now (clears R1)

; If modifier IO$M_DATAPATH is set,
; queue packet.  The data path is changed at driver level to preserve
; order with other requests.

20$:    BBS     S^#IO$V_DATAPATH,R0,30$ ; If BDP modifier set, queue packet

        JMP     G^EXE$SETCHAR           ; Set device characteristics

; This is a request to change data path useage, queue packet
```

```
30$:    CMPL    #IO$_SETCHAR,R7         ; Set characteristics?
        BNEQ    45$                     ; No, must have the privelege
        JMP     G^EXE$SETMODE           ; Queue packet to start I/O

; Error, abort IO

45$:    MOVZWL  #SS$_NOPRIV,R0          ; No priv for operation
50$:    CLRL    R1
        JMP     G^EXE$ABORTIO           ; Abort IO on error
        .SBTTL  XA_START, Start I/O routines
;++
; XA_START - Start a data transfer, set characteristics, enable ATTN AST.
;
; Functional Description:
;
;       This routine has two major functions:
;
;       1) Start an I/O transfer.  This transfer can be in either word
;          or block mode.  The FNCTN bits in the DR11-W CSR are set.  If
;          the transfer count is zero, the STATUS bits in the DR11-W CSR
;          are read and the request completed.
;       2) Set Characteristics.  If the function is change data path, the
;          new data path flag is set in the UCB.
;
; Inputs:
;
;       R3 = Address of the I/O request packet
;       R5 = Address of the UCB
;
; Outputs:
;
;       R0 = final status and number of bytes transferred
;       R1 = value of CSR STATUS bits and value of input data buffer register
;       Device errors are logged
;       Diagnostic buffer is filled
;
;--
        .ENABL  LSB

XA_START:

; Retrieve the address of the device CSR

        ASSUME  IDB$L_CSR EQ 0
        MOVL    UCB$L_CRB(R5),R4        ; Address of CRB
        MOVL    @CRB$L_INTD+VEC$L_IDB(R4),R4
                                        ; Address of CSR

; Fetch the I/O function code

        MOVZWL  IRP$W_FUNC(R3),R1       ; Get entire function code
        MOVW    R1,UCB$W_FUNC(R5)       ; Save FUNC in UCB for Error Logging
        EXTZV   #IO$V_FCODE,#IO$S_FCODE,R1,R2 ; Extract function field

; Dispatch on function code.  If this is SET CHARACTERISTICS, we will
; select a data path for future use.
; If this is a transfer function, it will either be processed in word
; or block mode.

        CMPB    #IO$_SETCHAR,R2         ; Set characteristics?
        BNEQ    3$

;++
; SET CHARACTERISTICS - Process Set Characteristics QIO function
;
```

```
; INPUTS:
;
;       XA_DATAPATH bit in Device Characteristics specifies which data path
;       to use.  If bit is a one, use buffered data path.  If zero, use
;       direct datapath.
;
; OUTPUTS:
;
;       CRB is flagged as to which datapath to use.
;       DEVDEPEND bits in device characteristics is updated
;               XA_DATAPATH = 1 -> buffered data path in use
;               XA_DATAPATH = 0 -> direct data path in use
;--

        MOVL    UCB$L_CRB(R5),R0                ; Get CRB address
        MOVQ    IRP$L_MEDIA(R3),UCB$B_DEVCLASS(R5) ; Set device characteristics
        BISB    #VEC$M_PATHLOCK,CRB$L_INTD+VEC$B_DATAPATH(R0)
                                                ; Assume direct datapath
        BBC     #XA$V_DATAPATH,UCB$L_DEVDEPEND(R5),2$ ;  Were we right?
        BICB    #VEC$M_PATHLOCK,CRB$L_INTD+VEC$B_DATAPATH(R0) ; Set buffered datapath
2$:
        CLRL    R1                              ; Return Success
        MOVZWL  #SS$_NORMAL,R0
        REQCOM

; If subfunction modifier for device reset is set, do one here

3$:     BBC     S^#IO$V_RESET,R1,4$     ; Branch if not device reset
        BSBW    XA_DEV_RESET            ; Reset DR11-W

; This must be a data transfer function - i.e. READ OR WRITE
; Check to see if this is a zero length transfer.
; If so, only set CSR FNCT bits and return STATUS from CSR

4$:     TSTW    UCB$W_BCNT(R5)          ; Is transfer count zero?
        BNEQ    10$                     ; No, continue with data transfer
        BBC     S^#IO$V_SETFNCT,R1,6$   ; Set CSR FNCT specified?
        DSBINT
        MOVW    IRP$L_SEGVBN+2(R3),XA_ODR(R4)
                                        ; Store word in ODR
        MOVZWL  XA_CSR(R4),R0
        BICW    #<XA_CSR$M_FNCT!XA_CSR$M_ERROR>,R0
        BISW    IRP$L_SEGVBN(R3),R0
        BISW    #XA_CSR$M_ATTN,R0       ; Force ATTN on to prevent lost interrupt
        MOVW    R0,XA_CSR(R4)
        BBC     #XA$V_LINK,UCB$L_DEVDEPEND(R5),5$ ; Link mode?
        BICW3   #XA$K_FNCT2,R0,XA_CSR(R4)       ; Make FNCT bit 2 a pulse
5$:
        ENBINT
6$:
        BSBW    XA_REGISTER             ; Fetch DR11-W registers
        BLBS    R0,7$                   ; If error, then log it
        JSB     G^ERL$DEVICERR          ; Log a device error
7$:     JSB     G^IOC$DIAGBUFILL        ; Fill diagnostic buffer if specified
        MOVL    UCB$W_XA_CSR(R5),R1     ; Return CSR and EIR in R1
        MOVZWL  UCB$W_XA_ERROR(R5),R0   ; Return status in R0
        BISB    #XA_CSR$M_IE,XA_CSR(R4) ; Enable device interrupts
        REQCOM                          ; Request done

; Build CSR image in R0 for later use in starting transfers

10$:
        MOVZWL  UCB$W_BCNT(R5),R0        ; Fetch byte count
        DIVL3   #2,R0,UCB$L_XA_DPR(R5)   ; Make byte count into word count
        ;
        ; Set up UCB$W_CSRTMP used for loading CSR later
        ;
        MOVZWL  XA_CSR(R4),R0
        BICW    #^C<XA_CSR$M_FNCT>,R0
        BISW    #XA_CSR$M_IE!XA_CSR$M_ATTN,R0    ; Set Interrupt Enable and ATTN
        BBC     S^#IO$V_SETFNCT,R1,20$   ; Set FNCT bits in CSR?
        BICW    #<XA_CSR$M_FNCT>,R0      ; Yes, Clear previous FNCT bits
        BISB    IRP$L_SEGVBN(R3),R0      ; OR in new value
20$:    BBC     S^#IO$V_DIAGNOSTIC,R1,23$        ; Check for maintenance function
        BISW    #XA_CSR$M_MAINT,R0       ; Set maintenance bit in CSR image

; Is this a word mode or block mode request?

23$:    MOVW    R0,UCB$W_XA_CSRTMP(R5)   ; Save CSR image in UCB
        BBC     S^#IO$V_WORD,R1,BLOCK_MODE ; Check if word or block mode
        BRW     WORD_MODE                ; Branch to handle word mode
```

```
;++
; BLOCK MODE -- Process a Block Mode (DMA) transfer request
;
; FUNCTIONAL DESCRIPTION:
;
;       This routine takes the buffer address, buffer size, fucntion code,
;       and function modifier fields from the IRP.  It calculates the UNIBUS
;       address, allocates the UBA map registers, loads the DR11-W device
;       registers and starts the request.
;--
; Set up UBA
; Start transfer

BLOCK_MODE:

; If IO$M_CYCLE subfunction is specified, set CYCLE bit in CSR image

        BBC     #IO$V_CYCLE,R1,25$      ; Set CYCLE bit in CSR?
        BISW    #XA_CSR$M_CYCLE,UCB$W_XA_CSRTMP(R5) ; If yes, or into CSR image

; Allocate UBA data path and map registers

25$:
        REQDPR                          ; Request UBA data path
        REQMPR                          ; Request UBA map registers
        LOADUBA                         ; Load UBA map registers

; Calculate the UNIBUS transfer address for the DR11-W from the UBA
; map register address and byte offset.

        MOVZWL  UCB$W_BOFF(R5),R1       ; Byte offset in first page of xfer
        MOVL    UCB$L_CRB(R5),R2        ; Address of CRB
        INSV    CRB$L_INTD+VEC$W_MAPREG(R2),#9,#9,R1
                                        ; Insert page number
        EXTZV   #16,#2,R1,R2            ; Extract bits 17:16 of bus address
        ASHL    #XA_CSR$V_XBA,R2,R2     ; Shift extended memeroy bits for CSR
        BISW    #XA_CSR$M_GO,R2         ; Set "GO" bit into CSR image
        BISW    R2,UCB$W_XA_CSRTMP(R5)  ; Set into CSR image we are building
        BICW3   #<XA_CSR$M_GO!XA_CSR$M_CYCLE>,UCB$W_XA_CSRTMP(R5),R0
                                        ; CSR image less "GO" and "CYCLE"
        BICW3   #XA$K_FNCT2,UCB$W_XA_CSRTMP(R5),R2 ; CSR image less FNCT bit 2
        MOVW    R1,UCB$W_XA_BARTMP(R5)  ; Save BAR for error logging

; At this juncture:
;       R0 = CSR image less "GO" and "CYCLE"
;       R1 = low 16 bits of transfer bus address
;       R2 = CSR image less FNCT bit 2
;       UCB$L_XA_DPR(R5) = transfer count in words
;       UCB$W_XA_CSRTMP(R5) = CSR image to start transfer with

; Set DR11-W registers and start transfer
; Note that read-modify-write cycles are NOT performed to the DR11-W CSR.
; The CSR is always written directly into.  This prevents inadvertently setting
; the EIR select flag (writing bit 15) if error happens to become true.

        DSBINT                          ; Disable interrupts (powerfail)
        MNEGW   UCB$L_XA_DPR(R5),XA_WCR(R4)
                                        ; Load negative of transfer count
        MOVW    R1,XA_BAR(R4)           ; Load low 16 bits of bus address
        MOVW    R0,XA_CSR(R4)           ; Load CSR image less "GO" and "CYCLE"
        BBC     #XA$V_LINK,UCB$L_DEVDEPEND(R5),26$ ; Link mode?
        MOVW    R2,XA_CSR(R4)           ; Yes, load CSR image less "FNCT" bit 2
        BRB     126$            ; Only if link mode in dev characteristics
26$:
        MOVW    UCB$W_XA_CSRTMP(R5),XA_CSR(R4) ; Move all bits to CSR

; Wait for transfer complete interrupt, powerfail, or device time-out

126$:
        WFIKPCH XA_TIME_OUT,IRP$L_MEDIA(R3) ; Wait for interrupt

; Device has interrupted, FORK

        IOFORK                          ; FORK to lower IPL

; Handle request completion, release UBA resources, check for errors

        MOVZWL  #SS$_NORMAL,-(SP)       ; Assume success, store code on stack
        CLRW    UCB$Q_XA_DPRN(R5)       ; Clear DPR number and DPR error flag
        PURDPR                          ; Purge UBA buffered data path
        BLBS    R0,27$                  ; Branch if no datapath error
        MOVZWL  #SS$_PARITY,(SP)        ; Flag parity error on device
        INCB    UCB$Q_XA_DPRN+1(R5)     ; Flag PDR error for log
27$:    MOVL    R1,UCB$L_XA_DPR(R5)     ; Save data path register in UCB
        EXTZV   #VEC$V_DATAPATH,-       ; Get Datapath register no.
                #VEC$S_DATAPATH,-       ; For Error Log
                CRB$L_INTD+VEC$B_DATAPATH(R3),R0
        MOVB    R0,UCB$W_XA_DPRN(R5)    ; Save for later in UCB
        EXTZV   #9,#7,UCB$W_XA_BAR(R5),R0 ; Low bits, final map register no.
        EXTZV   #4,#2,UCB$W_XA_CSR(R5),R1 ; Hi bits of map register no.
        INSV    R1,#7,#2,R0             ; Entire map register number
        CMPW    R0,#496                 ; Is map register number in range?
        BGTR    28$                     ; No, forget it - compound error
        MOVL    (R2)[R0],UCB$L_XA_FMPR(R5) ; Save map register contents
        CLRL    UCB$L_XA_PMPR(R5)       ; Assume no previous map register
        DECL    R0                      ; Was there a previous map register?
        CMPV    #VEC$V_MAPREG,#VEC$S_MAPREG,-
                CRB$L_INTD+VEC$W_MAPREG(R3),R0
        BGTR    28$                     ; No if gtr
        MOVL    (R2)[R0],UCB$L_XA_FMPR(R5) ; Save previous map register contents
28$:    RELMPR                          ; Release UBA resources
        RELDPR

; Check for errors and return status

        TSTW    UCB$W_XA_WCR(R5)        ; All words transferred?
        BEQL    30$                     ; Yes
        MOVZWL  #SS$_OPINCOMPL,(SP)     ; No, flag operation not complete
30$:    BBC     #XA_CSR$V_ERROR,UCB$W_XA_CSR(R5),35$ ; Branch on CSR error bit
        MOVZWL  UCB$W_XA_ERROR(R5),(SP) ; Flag for controller/drive error status
        BSBW    XA_DEV_RESET            ; Reset DR11-W
35$:    BLBS    (SP),40$                ; Any errors after all this?
```

```
        JSB     G^ERL$DEVICERR          ; Yes, log them
40$:    BSBW    DEL_ATTNAST             ; Deliver outstanding ATTN AST's
        JSB     G^IOC$DIAGBUFILL        ; Fill diagnostic buffer
        MOVL    (SP)+,R0                ; Get final device status
        MULW3   #2,UCB$W_XA_WCR(R5),R1  ; Calculate final transfer count
        ADDW    UCB$W_BCNT(R5),R1
        INSV    R1,#16,#16,R0           ; Insert into high byte of IOSB
        MOVL    UCB$W_XA_CSR(R5),R1     ; Return CSR and EIR in IOSB
        BISB    #XA_CSR$M_IE,XA_CSR(R4) ; Enable interrupts
        REQCOM                          ; Finish request in exec
        .DSABL  LSB
;++
; WORD MODE -- Process word mode (interrupt per word) transfer
;
; FUNCTIONAL DESCRIPTION:
;
;       Data is transferred one word at a time with an interrupt for each word.
;       The request is handled separately for a write (from memory to DR11-W
;       and a read (from DR11-W to memory).
;       For a write, data is fetched from memory, loaded into the ODR of the
;       DR11-W and the system waits for an interrupt.  For a read, the system
;       waits for a DR11-W interrupt and the IDR is transferred into memory.
;       If the unsolicited interrupt flag is set, the first word is transferred
;       directly into memory withou waiting for an interrupt.
;--

        .ENABL  LSB
WORD_MODE:

; Dispatch to separate loops on READ or WRITE

        CMPB    #IO$_READPBLK,R2        ; Check for read function
        BEQL    30$

;++
; WORD MODE WRITE -- Write (output) in word mode
;
; FUNCTIONAL DESCRIPTION:
;
;       Transfer the requested number of words from user memory to
;       the DR11-W ODR one word at a time, wait for interrupt for each
;       word.
;--

10$:
        BSBW    MOVFRUSER                       ; Get two bytes from user buffer
        DSBINT                                  ; Lock out interrupts
                                                ; Flag interrupt expected
        MOVW    R1,XA_ODR(R4)                   ; Move data to DR11-W
        MOVW    UCB$W_XA_CSRTMP(R5),XA_CSR(R4)  ; Set DR11-W CSR
        BBC     #XA$V_LINK,UCB$L_DEVDEPEND(R5),15$ ; Link mode?
        BICW3   #XA$K_FNCT2,UCB$B_XA_CSRTMP(R5),XA_CSR(R4) ; Clear interrupt FNCT bit 2
                                                ; Only if link mode specified
15$:

; Wait for interrupt, powerfail, or device time-out

        WFIKPCH XA_TIME_OUTW,IRP$L_MEDIA(R3)

; Check for errors, decrement transfer count, and loop til complete

        IOFORK                                  ; Fork to lower IPL
        BITW    #XA_EIR$M_NEX!-
                XA_EIR$M_MULTI!-
                XA_EIR$M_ACLO!-
                XA_EIR$M_PAR!-
                XA_EIR$M_DLT,UCB$W_XA_EIR(R5) ; Any errors?
        BEQL    20$                           ; No, continue
        BRW     40$                           ; Yes, abort transfer.
20$:    DECW    UCB$L_XA_DPR(R5)                      ; All words trnasferred?
        BNEQ    10$                           ; No, loop until finished.

; Transfer is done, clear iterrupt expected flag and FORK
; All words read or written in WORD MODE.  Finish I/O.

RETURN_STATUS:

        JSB     G^IOC$DIAGBUFILL              ; Fill diagnostic buffer if present
        BSBW    DEL_ATTNAST                   ; Deliver outstanding ATTN AST's
        MOVZWL  #SS$_NORMAL,R0                ; Complete success status
22$:    MULW3   #2,UCB$L_XA_DPR(R5),R1        ; Calculate actual bytes xfered
        SUBW3   R1,UCB$W_BCNT(R5),R1          ; From requested number of bytes
        INSV    R1,#16,#16,R0                 ; And place in high word of R0
        MOVL    UCB$W_XA_CSR(R5),R1           ; Return CSR and EIR status
        BISB    #XA_CSR$M_IE,XA_CSR(R4)       ; Enable device interrupts
        REQCOM                                ; Finish request in exec

;++
; WORD MODE READ -- Read (input) in word mode
;
; FUNCTIONAL DESCRIPTION:
;
;       Transfer the requested number of wrods from the DR11-W IDR into
;       user memory one word at a time, wait for interrupt for each word.
;       If the unexpected (unsolicited) interrupt bit is set, transfer the
;       first (last received) word to memory without waiting for an
;       interrupt.
;--

30$:
        DSBINT  UCB$B_DIPL(R5)                ; Lock out interrupts

; If an unexpected (unsolicited) interrupt has occured, assume it
; is for this READ request and return value to user buffer without
; waiting for an interrupt.

        BBCC    #UCB$V_UNEXPT,-
                UCB$W_DEVSTS(R5),32$          ; Branch if no unexpected interrupt
        ENBINT                                ; Enable interrupts
        BRB     37$                           ; continue

32$:
        SETIPL  #IPL$_POWER
35$:

; Wait for interrupt, powerfail, or device time-out

        WFIKPCH XA_TIME_OUTW,IRP$L_MEDIA(R3)

; Check for errors, decrement transfer count and loop until done
```

```
        IOFORK                          ; Fork to lower IPL
37$:
        BITW    #XA_EIR$M_NEX!-
                XA_EIR$M_MULTI!-
                XA_EIR$M_ACLO!-
                XA_EIR$M_PAR!-
                XA_EIR$M_DLT,UCB$W_XA_EIR(R5) ; Any errors?
        BNEQ    40$                     ; Yes, abort transfer.
        BSBW    MOVTOUSER               ; Store two bytes into user buffer

; Send interrupt back to sender.  Acknowledge we got last word.

        DSBINT
        MOVW    UCB$W_XA_CSRTMP(R5),XA_CSR(R4)
        BBC     #XA$V_LINK,UCB$L_DEVDEPEND(R5),38$ ; Link mode?
        BICW3   #XA$K_FNCT2,UCB$W_XA_CSRTMP(R5),XA_CSR(R4) ; Yes, clear FNCT 2
38$:
        DECW    UCB$L_XA_DPR(R5)                ; Decrement transfer count
        BNEQ    35$                     ; Loop until all words transferred
        ENBINT
        BRW     RETURN_STATUS           ; Finish request in common code

; Error detected in word mode transfer

40$:
        BSBW    DEL_ATTNAST             ; Deliver ATTN AST's
        BSBW    XA_DEV_RESET            ; Error, reset DR11-W
        JSB     G^IOC$DIAGBUFILL        ; Fill diagnostic buffer if presetn
        JSB     G^ERL$DEVICERR          ; Log device error
        MOVZWL  UCB$W_XA_ERROR(R5),R0   ; Set controller/drive status in R0
        BRW     22$

        .DSABL  LSB
;
; MOVFRUSER - Routine to fetch two bytes from user buffer.
;
; INPUTS:
;
;       R5 = UCB address
;
; OUTPUTS:
;
;       R1 = Two bytes of data from users buffer
;       Buffer descriptor in UCB is updated.
;
        .ENABL  LSB
MOVFRUSER:
        MOVAL   -(SP),R1                ; Address of temporary stack loc
        MOVZBL  #2,R2                   ; Fetch two bytes
        JSB     G^IOC$MOVFRUSER         ; Call exec routine to do the deed
        MOVL    (SP)+,R1                ; Retreive the bytes
        BRB     20$                     ; Update UCB buffer pointers
;
; MOVTOUSER - Routine to store two bytes into users buffer.
;
```

```
; INPUTS:
;
;       R5 = UCB address
;       UCB$W_XA_IDR(R5) = Location where two bytes are saved
;
; OUTPUTS:
;
;       Two bytes are stored in user buffer and buffer descriptor in
;       UCB is updated.
;
MOVTOUSER:
        MOVAB   UCB$W_XA_IDR(R5),R1     ; Address of internal buffer
        MOVZBL  #2,R2
        JSB     G^IOC$MOVTOUSER         ; Call exec
20$:                                    ; Update buffer pointers in UCB
        ADDW    #2,UCB$W_BOFF(R5)       ; Add two to buffer descriptor
        BICW    #^C<^X01FF>,UCB$W_BOFF(R5) ; Modulo the page size
        BNEQ    30$                     ; If NEQ, no page boundary crossed
        ADDL    #4,UCB$L_SVAPTE(R5)     ; Point to next page
30$:
        RSB
;
        .DSABL  LSB
        .PAGE
        .SBTTL  DR11-W DEVICE TIME-OUT
;++
; DR11-W device TIME-OUT
; If a DMA transfer was in progress, release UBA resources.
; For DMA or WORD mode, deliver ATTN AST's, log a device timeout error,
; and do a hard reset on the controller.
;
; Clear DR11-W CSR
; Return error status
;
; Power failure will appear as a device time-out
;--
        .ENABL  LSB
XA_TIME_OUT:                            ; Time-out for DMA transfer

        SETIPL  UCB$B_FIPL(R5)          ; Lower to FORK IPL
        PURDPR                          ; Purge buffered data path in UBA
        RELMPR                          ; Release UBA map registers
        RELDPR                          ; Release UBA data path
        BRB     10$                     ; continue

XA_TIME_OUTW:                           ; Time-out for WORD mode transfer

        SETIPL  UCB$B_FIPL(R5)          ; Lower to FORK IPL
10$:    MOVL    UCB$L_CRB(R5),R4        ; Fetch address of CSR
        MOVL    @CRB$L_INTD+VEC$L_IDB(R4),R4
        BSBW    XA_REGISTER             ; Read DR11-W registers
        JSB     G^IOC$DIAGBUFILL        ; Fill diagnostic buffer
        JSB     G^ERL$DEVICTMO          ; Log device time out
        BSBW    DEL_ATTNAST             ; And deliver the AST's
        BSBW    XA_DEV_RESET            ; Reset controller
        MOVZWL  #SS$_TIMEOUT,R0         ; Assume error status
        BBC     #UCB$V_CANCEL,-
                UCB$W_STS(R5),20$       ; Branch if not cancel
        MOVZWL  #SS$_CANCEL,R0          ; Set status
20$:    CLRL    R1
        CLRW    UCB$W_DEVSTS(R5)        ; Clear ATTN AST flags
        BICW    #<UCB$M_TIM!UCB$M_INT!UCB$M_TIMOUT!UCB$M_CANCEL!UCB$M_POWER>,-
                UCB$W_STS(R5)           ; Clear unit status flags
        REQCOM                          ; Complete I/O in exec
        .DSABL  LSB
        .PAGE
        .SBTTL  XA_INTERRUPT, Interrupt service routine for DR11-W
;++
; XA_INTERRUPT, Handles interrupts generated by DR11-W
;
; Functional description:
;
;       This routine is entered whenever an interrupt is generated
;       by the DR11-W.  It checks that an interrupt was expected.
;       If not, it sets the unexpected (unsolicited) interrupt flag.
;       All device registers are read and stored into the UCB.
;       If an interrupt was expected, it calls the driver back at its Wait
;       for Interrupt point.
;       Deliver ATTN AST's if unexpected interrupt.
;
; Inputs:
;
;       00(SP) = Pointer to address of the device IDB
;       04(SP) = saved R0
;       08(SP) = saved R1
;       12(SP) = saved R2
;       16(SP) = saved R3
;       20(SP) = saved R4
;       24(SP) = saved R5
;       28(SP) = saved PSL
;       32(SP) = saved PC
;
; Outputs:
;
;       The driver is called at its Wait For Interrupt point if an
;       interrupt was expected.
;       The current value of the DR11-W CSR's are stored in the UCB.
;
;--
XA_INTERRUPT:                           ; Interrupt service for DR11-W
        MOVL    @(SP)+,R4               ; Address of IDB and pop SP
        MOVQ    (R4),R4                 ; CSR and UCB address from IDB

; Read the DR11-W device registers (WCR, BAR, CSR, EIR, IDR) and store
; into UCB.

        BSBW    XA_REGISTER             ; Read device registers

; Check to see if device transfer request active or not
; If so, call driver back at Wait for Interrupt point and
; Clear unexpected interrupt flag.

20$:    BBCC    #UCB$V_INT,UCB$W_STS(R5),25$
                                        ; If clear, no interrupt expected

; Interrupt expected, clear unexpected interrupt flag and call driver
; back.

        BICW    #UCB$M_UNEXPT,UCB$W_DEVSTS(R5)
                                        ; Clear unexpected interrupt flag
        MOVL    UCB$L_FR3(R5),R3        ; Restore drivers R3
        JSB     @UCB$L_FPC(R5)          ; Call driver back
        BRB     30$

; Deliver ATTN AST's if no interrupt expected and set unexpected
; interrupt flag.

25$:
        BISW    #UCB$M_UNEXPT,UCB$W_DEVSTS(R5) ; Set unexpected interrupt flag
        BSBW    DEL_ATTNAST             ; Deliver ATTN AST's
        BISB    #XA_CSR$M_IE,XA_CSR(R4) ; Enable device interrupts

; Restore registers and return from interrupt

30$:
        POPR    #^M<R0,R1,R2,R3,R4,R5>  ; Restore registers
        REI                             ; Return from interrupt
        .PAGE
        .SBTTL  XA_REGISTER - Handle DR11-W CSR transfers
;++
; XA_REGISTER - Routine to handle DR11-W register transfers
;
; INPUTS:
;
;       R4 - DR11-W CSR address
;       R5 - UCB address of unit
;
; OUTPUTS:
;
;       CSR, EIR, WCR, BAR, IDR, and status are read and stored into UCB.
;       The DR11-W is placed in its initial state with interrupts enabled.
;       R0 - .true. if no hard error
;            .false. if hard error (cannot clear ATTN)
;
; If the CSR ERROR bit is set and the associated condition can be cleared, then
; the error is transient and recoverable.  The status returned is SS$_DRVERR.
; If the CSR ERROR bit is set and cannot be cleared by clearing the CSR, then
; this is a hard error and cannot be recovered.  The returned status is
; SS$_CTRLERR.
;
;       R0,R1 - destroyed, all other registers preserved.
;--

XA_REGISTER:

        MOVZWL  #SS$_NORMAL,R0          ; Assume success
        MOVZWL  XA_CSR(R4),R1           ; Read CSR
        MOVW    R1,UCB$W_XA_CSR(R5)     ; Save CSR in UCB
        BBC     #XA_CSR$V_ERROR,R1,55$  ; Branch if no error
        MOVZWL  #SS$_DRVERR,R0          ; Assume "drive" error
55$:    BICW    #^C<XA_CSR$M_FNCT>,R1   ; Clear all uninteresting bits for later
        BISB    #<XA_CSR$M_ERROR/256>,XA_CSR+1(R4) ; Set EIR flag
        MOVW    XA_EIR(R4),UCB$W_XA_EIR(R5) ; Save EIR in UCB
        MOVW    R1,XA_CSR(R4)           ; Clear EIR flag and errors
        MOVW    XA_CSR(R4),R1           ; Read CSR back
        BBC     #XA_CSR$V_ATTN,R1,60$   ; If attention still set, hard error
        MOVZWL  #SS$_CTRLERR,R0         ; Flag hard controller error
60$:    MOVW    XA_IDR(R4),UCB$W_XA_IDR(R5) ; Save IDR in UCB
        MOVW    XA_BAR(R4),UCB$W_XA_BAR(R5)
        MOVW    XA_WCR(R4),UCB$W_XA_WCR(R5)
        MOVW    R0,UCB$W_XA_ERROR(R5)   ; Save status in UCB
        RSB
        .SBTTL  XA_CANCEL, Cancel I/O routine
;++
; XA_CANCEL, Cancels an I/O operation in progress
;
; Functional description:
;
;       Flushes Attention AST queue for the user.
;       If transfer in progress, do a device reset to DR11-W and finish the
;       request.
;       Clear interrupt expected flag.
;
; Inputs:
;
;       R2 = negated value of channel index
;       R3 = address of current IRP
;       R4 = address of the PCB requesting the cancel
;       R5 = address of the device's UCB
;
; Outputs:
;
;--

XA_CANCEL:                                      ; Cancel I/O

        BBCC    #UCB$V_ATTNAST,-
                UCB$W_DEVSTS(R5),20$    ; ATTN AST enabled?

; Finish all ATTN AST's for this process.

        PUSHR   #^M<R2,R6,R7>
        MOVL    R2,R6                   ; Set up channel number
        MOVAB   UCB$L_XA_ATTN(R5),R7    ; Address of listhead
        JSB     G^COM$FLUSHATTNS        ; Flush ATTN AST's for process
        POPR    #^M<R2,R6,R7>

; Check to see if a data transfer request is in progress
; for this process on this channel

20$:
        DSBINT  UCB$B_DIPL(R5)          ; Lock out device interrupts
        JSB     G^IOC$CANCELIO          ; Check if transfer going
        BBC     #UCB$V_CANCEL,-
                UCB$W_STS(R5),30$       ; Branch if not for this guy
;
; Force timeout
;
        CLRL    UCB$L_DUETIM(R5)        ; clear timer
        BISW    #UCB$M_TIM,UCB$W_STS(R5) ; set timed
        BICW    #UCB$M_TIMOUT,-
                UCB$W_STS(R5)           ; Clear timed out
30$:
        ENBINT                          ; Lower to FORK IPL
        RSB                             ; Return
        .PAGE
        .SBTTL  DEL_ATTNAST, Deliver ATTN AST's
;++
; DEL_ATTNAST, Deliver all outstanding ATTN AST's
;
; Functional description:
;
;       This routine is used by the DR11-W driver to deliver all of the
;       outstanding attention AST's.  It is copied from COM$DELATTNAST in
;       the exec.  In addition, it places the saved value of the DR11-W CSR
;       and Input Data Buffer Register in the AST paramater.
;
; Inputs:
;
;       R5 = UCB of DR11-W unit
;
; Outputs:
;
;       R0,R1,R2 Destroyed
;       R3,R4,R5 Preserved
;--
DEL_ATTNAST:
        DSBINT  UCB$B_DIPL(R5)          ; Device IPL
        BBCC    #UCB$V_ATTNAST,UCB$W_DEVSTS(R5),30$
                                        ; Any ATTN AST's expected?
        PUSHR   #^M<R3,R4,R5>           ; Save R3,R4,R5
10$:    MOVL    8(SP),R1                ; Get address of UCB
        MOVAB   UCB$L_XA_ATTN(R1),R2    ; Address of ATTN AST listhead
        MOVL    (R2),R5                 ; Address of next entry on list
        BEQL    20$                     ; No next entry, end of loop
        BICW    #UCB$M_UNEXPT,UCB$W_DEVSTS(R1) ; Clear unexpected interrupt flag
        MOVL    (R5),(R2)               ; Close list
        MOVW    UCB$W_XA_IDR(R1),ACB$L_KAST+6(R5)
                                        ; Store IDR in AST paramater
        MOVW    UCB$W_XA_CSR(R1),ACB$L_KAST+4(R5)
                                        ; Store CSR in AST paramater
        PUSHAB  B^10$                   ; Set return address for FORK
        FORK                            ; FORK for this AST

; AST fork procedure

        MOVQ    ACB$L_KAST(R5),ACB$L_AST(R5)
                                        ; Re-arrange entries
        MOVB    ACB$L_KAST+8(R5),ACB$B_RMOD(R5)
        MOVL    ACB$L_KAST+12(R5),ACB$L_PID(R5)
        CLRL    ACB$L_KAST(R5)
        MOVZBL  #PRI$_IOCOM,R2          ; Set up priority increment
        JMP     G^SCH$QAST              ; Queue the AST

20$:    POPR    #^M<R3,R4,R5>           ; Restore registers
30$:    ENBINT                          ; Enable interrupts
        RSB                             ; Return
        .PAGE
        .SBTTL  XA_REGDUMP - DR11-W register dump routine
;++
; XA_REGDUMP - DR11-W Register dump routine.
;
; This routine is called to save the controller registers in a specified
; buffer.  It is called from the device error logging routine and from the
; diagnostic buffer fill routine.
;
; Inputs:
;
;       R0 - Address of register save buffer
;       R4 - Address of Control and Status Register
;       R5 - Address of UCB
;
; Outputs:
;
;       The controller registers are saved in the specified buffer.
;
;               CSRTMP - The last command written to the DR11-W CSR by
;                        by the driver.
;               BARTMP - The last value written into the DR11-W BAR by
;                        the driver during a block mode transfer.
;               CSR - The CSR image at the last interrupt
;               EIR - The EIR image at the last interrupt
;               IDR - The IDR image at the last interrupt
;               BAR - The BAR image at the last interrupt
;               WCR - Word count register
;               ERROR - The system status at request completion
;               PDRN - UBA Datapath Register number
;               DPR - The contents of the UBA Data Path register
;               FMPR - The contents of the last UBA Map register
;               PMRP - The contents of the previous UBA Map register
;               DPRF - Flag for purge datapath error
;                       0 = no purger datapath error
;                       1 = parity error when datapath was purged
;
;       Note that the values stored are from the last completed transfer
;       operation. If a zero transfer count is specified, then the
;       values are from the last operation with a non-zero transfer count.
;--

XA_REGDUMP:

        MOVZBL  #11,(R0)+               ; Eleven registers are stored.
        MOVAB   UCB$W_XA_CSRTMP(R5),R1  ; Get address of saved register images
        MOVZBL  #8,R2                   ; Return 8 registers here
10$:    MOVZWL  (R1)+,(R0)+
        SOBGTR  R2,10$                  ; Move them all
        MOVZBL  UCB$W_XA_DPRN(R5),(R0)+ ; Save Datapath Register number
        MOVZBL  #3,R2                   ; And 3 more here
20$:    MOVL    (R1)+,(R0)+             ; Move UBA register contents
        SOBGTR  R2,20$
        MOVZBL  UCB$W_XA_DPRN+1(R5),(R0)+ ; Save Datapath Parity Error Flag
        RSB
        .PAGE
        .SBTTL  XA_DEV_RESET - Device reset DR11-W
;++
; XA_DEV_RESET - DR11-W Device reset routine
;
; This routine raises IPL to device IPL, performs a device reset to
; the required controler, and re-enables device interrupts.
;
; Inputs:
;
;       R4 - Address of Control and Status Register
;       R5 - Address of UCB
;
; Outputs:
;
;       Controller is reset, controller interrupts are enabled
;
;--

XA_DEV_RESET:

        PUSHR   #^M<R0,R1,R2>              ; Save some registers
        DSBINT                             ; Raise IPL to lock all interrupts
        MOVB    #<XA_CSR$M_MAINT/256>,XA_CSR+1(R4)
        CLRB    XA_CSR+1(R4)

; ***  Must delay here depending on reset interval

        TIMEDWAIT TIME=#XA_RESET_DELAY  ; No. of 10 micro-sec intervals to wait

        MOVB    #XA_CSR$M_IE,XA_CSR(R4) ; Re-enable device interrupts
        ENBINT                             ; Restore IPL
        POPR    #^M<R0,R1,R2>              ; Restore registers

        RSB

XA_END:                                            ; End of driver label
        .END
```

0157 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

LPMULT
B32

DRMAST
MAR

ADDRIVER
MAR

TDRIVER
MAR

USSTEST
MAR

GBLSECUFO
MAR

USSDISP
MAR

DOD_ERAPAT
MAR

LBRMAC
MAR

XADRIVER
MAR

LABIOCIN
MAR

DRSLV
MAR

DTE_DF03
MAR

WORKQ
LIS

SCRFT
MAR

EXAMPLES